JSPM’s

Bhivarabai Sawant Institute of Technology and Research

(Approved by AICTE, Delhi & Govt. of Maharashtra, affiliated to Savitribai Phule Pune University)

Gat. No.720, Pune-Nagar Road, Wagholi, Pune-412207

Department of Information Technology

Computer graphics Lab
Subject Code: 214453

A.Y.2021-22

SE IT (2019Course)

Semester II

LABORATORY MANUAL

Subject Incharge:-Prof.Bhagyashree Kadam

Vision of Institute

" To Satisfy the aspirations of youth force, who wants to lead nation towards prosperity through techno-economic developments "

Mission of Institute

" To provide, nurture and maintain an environment of high academic excellence, research and entrepreneurship for all aspiring students which will prepare them to face global challenges, maintaining high ethical and moral standard "

Vision of Department

To be a nucleus nurturing learner to cater current & future digital needs

Mission of Department

1. Educating aspirants to fulfill technological and social needs through effective teaching learning process.

2. Imparting IT skills to develop innovative solutions catering needs of multidisciplinary domain.

DEPARTMENT OF INFORMATION TECHNOLOGY

SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE

PROGRAM OUTCOME (PO)

Sr.No.	PO	Description
PO1	Engineering knowledge	An ability to apply knowledge of mathematics, computing, science, engineering and technology.
PO2	Problem analysis	An ability to define a problem and provide a systematic solution with the help of conducting experiments, analyzing the problem and interpreting the data.
PO3	Design / Development of Solutions	An ability to design, implement, and evaluate a software or a software/hardware system, component, or process to meet desired needs within realistic constraints
PO4	Conduct Investigations of Complex Problems	An ability to identify, formulates, and provides systematic solutions to complex engineering/Technology problems.
PO5	Modern Tool Usage	An ability to use the techniques, skills, and modern engineering technology tools, standard processes necessary for practice as a IT professional.
PO6	The Engineer and Society	An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-basedsystems with necessary constraints and assumptions.
PO7	Environment and Sustainability	An ability to analyze and provide solution for the local and global impact of information technology on individuals, organizations and society
PO8	Ethics	An ability to understand professional, ethical, legal, security and social issues and responsibilities.
PO9	Individual and Team Work	An ability to function effectively as an individual or as a team member to accomplish a desired goal(s).
PO10	Communication Skills	An ability to engage in life-long learning and continuing professional development to cope up with fast changes in the technologies/tools with the help of electives, professional organizations and extracurricular activities
PO11	Project Management and Finance	An ability to communicate effectively in engineering community at large by means of effective presentations, report writing, paper publications, demonstrations.
PO12	Life-long Learning	An ability to understand engineering, management, financial aspects, performance, optimizations and time complexity necessary for professional practice

PROGRAM SPECIFIC OUTCOME (PSO)

PSO1	An ability to apply the theoretical concepts and practical knowledge of Information Technology in analysis, design, development and management of information processing systems and applications in the interdisciplinary domain.
PSO2	An ability to analyze a problem, and identify and define the computing infrastructure and operations requirements appropriate to its solution. IT graduates should be able to work on large-scale computing systems.
PSO3	An understanding of professional, business and business processes, ethical, legal, security and social issues and responsibilities
PSO4	Practice communication and decision-making skills through the use of appropriate technology and be ready for professional responsibilities.

CO1	Apply mathematical and logical aspects for developing elementary graphics operations like scan conversion of points, lines, circle, and apply it for problem solving.
CO2	Employ techniques of geometrical transforms to produce, position and manipulate Objects in 2 dimensional and 3-dimensional space respectively.
CO3	Describe mapping from a world coordinates to device coordinates, clipping, and projections in order to produce 3D images on 2D output device.
CO4	Apply concepts of rendering, shading, animation, curves and fractals using computer graphics tools in design, development and testing of 2D, 3D modeling applications
CO5	Perceive the concepts of virtual reality.

214453**:** Computer graphics Lab

Teaching Scheme:	Practical: 2Hours/Week	**Credits: 02**
Examination Scheme:	Practical: 25 Marks	

Prerequisites: Basic Geometry, Trigonometry, Vectors and Matrices, Data Structures and Algorithms

Sr. No	**List of Assignments**	**Page No**
1	Install and explore the OpenGL	6
2	Implement DDA and Bresenham line drawing algorithm to draw: i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line ;using mouse interface Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes positive as well as negative.	19
	Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius	
4	Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming	30
	Implement Cohen Sutherland polygon clipping method to clip the polygon with respect the viewport and window. Use mouse click, keyboard interface	
6	Implement following 2D transformations on the object with respect to axis : – CO5 i) Scaling ii) Rotation about arbitrary point iii) Reflection	42
7	a) Generate fractal patterns using i) Bezier ii) Koch Curve -	
8	Implement animation principles for any object	52

Assignment No. 1

Problem Statement:

Install and explore the OpenGL.

Aim:

Install and explore the OpenGL.

Theory:

Introduction to OpenGL-

The GL Library in OpenGL is the core library, Glu is a utility library, glut is a utility library, GL is the core, Glu is part of the package for GL, glut is OpenGL's cross-platform tool Library, GL contains the most basic 3D functions, and Glu seems to support GL, if the arithmetic is good, The same effect can be done without the glu of the case. Glut is the basic window interface, is independent of GL and Glu, if you do not like to use glut can use MFC and Win32 window, etc., but glut is cross-platform, which ensures that our program is a cross-platform, If you use MFC or WIN32 only on Windows operating systems. One of the big reasons for choosing OpenGL is that it's cross-platform, so we can use the glut library as much as possible.

1)glutInit

glutInit is used to initialize the GLUT library.

Usage

void glutInit(int *argcp, char **argv);

2)argcp

A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

3)argv

The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

4)glutInitDisplayMode

glutInitDisplayMode sets the initial display mode.

Usage

void glutInitDisplayMode(unsigned int mode);

mode

Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. See values below:

5)GLUT_RGBA

Bit mask to select an RGBA mode window. This is the default if neither GLUT_RGBA nor GLUT_INDEX are specified.

6)GLUT_RGB

An alias for GLUT_RGBA.

7)GLUT_INDEX

Bit mask to select a color index mode window. This overrides GLUT_RGBA if it is also specified.

8)GLUT_SINGLE

Bit mask to select a single buffered window. This is the default if neither GLUT_DOUBLE or GLUT_SINGLE are specified.

9)GLUT_DOUBLE

Bit mask to select a double buffered window. This overrides GLUT_SINGLE if it is also specified.

10)GLUT_ACCUM

Bit mask to select a window with an accumulation buffer.

11)GLUT_ALPHA

Bit mask to select a window with an alpha component to the color buffer(s).

12)GLUT_DEPTH

Bit mask to select a window with a depth buffer.

13)GLUT_STENCIL

Bit mask to select a window with a stencil buffer.

14)GLUT_MULTISAMPLE

Bit mask to select a window with multisampling support. If multisampling is not available, a non-multisampling window will automatically be chosen. Note: both the OpenGL client-side and server-side implementations must support the GLX_SAMPLE_SGIS extension for multisampling to be available.

15)GLUT_STEREO

Bit mask to select a stereo window.

16)glutInitWindowPosition, glutInitWindowSize

glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.

Usage

```
void glutInitWindowSize(int width, int height);

void glutInitWindowPosition(int x, int y);
```

width - Width in pixels.

Height - Height in pixels.

X - Window X location in pixels.

Y - Window Y location in pixels.

17)glutCreateWindow

glutCreateWindow creates a top-level window.

Usage

```
int glutCreateWindow(char *name);
```

name - ASCII character string for use as window name.

18)glutMouseFunc

glutMouseFunc sets the mouse callback for the current window.

Usage

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
```

func

The new mouse callback function.

Description-

glutMouseFunc sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback.

19)glutDisplayFunc

glutDisplayFunc sets the display callback for the current window.

Usage

void glutDisplayFunc(void (*func)(void));

func

The new display callback function.

Description-

glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called.

20)glutMainLoop

glutMainLoop enters the GLUT event processing loop.

Usage

void glutMainLoop(void);

Description-

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

21)glClear(GL_COLOR_BUFFER_BIT);

glClear — clear buffers to preset values

GL_COLOR_BUFFER_BIT

Indicates the buffers currently enabled for color writing.

22)glClearColor — specify clear values for the color buffers

C Specification void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha); [0,1]

23)gluOrtho2D — define a 2D orthographic projection matrix

C Specification void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top); left, right specify the coordinates for the left and right vertical clipping planes. bottom, top specify the coordinates for the bottom and top horizontal clipping planes.

24)glReadPixels — read a block of pixels from the frame buffer

C Specification void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid * data); Parameters x, y Specify the window coordinates of the first pixel that is read from the frame buffer.

This location is the lower left corner of a rectangular block of pixels. width, height specify the dimensions of the pixel rectangle. width and height of one correspond to a single pixel format specifies the format of the pixel data. The following symbolic values are accepted:

GL_ALPHA,

GL_RGB, and

GL_RGBA.

Type-

Specifies the data type of the pixel data. Must be one of

GL_UNSIGNED_BYTE,

GL_UNSIGNED_SHORT_5_6_5,

GL_UNSIGNED_SHORT_4_4_4_4, or

GL_UNSIGNED_SHORT_5_5_5_1.

Data-

Returns the pixel data.

25)glutIdleFunc

glutIdleFunc sets the global idle callback.

Usage

void glutIdleFunc(void (*func)(void));

Description-

glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received. If enabled, the idle callback is continuously called when events are not being received.

26)glMatrixMode — specify which matrix is the current matrix

C Specification void glMatrixMode(GLenum mode); Parametersmode Specifies which matrix stack is the target for subsequent matrix operations.

Three values are accepted:

GL_MODELVIEW,

GL_PROJECTION, and

GL_TEXTURE.

The initial value is GL_MODELVIEW.

Additionally, if the ARB_imaging extension is supported,

GL_COLOR is also accepted.

Description

glMatrixMode sets the current matrix mode. mode can assume one of four values:

GL_MODELVIEW

Applies subsequent matrix operations to the modelview matrix stack.

GL_PROJECTION

Applies subsequent matrix operations to the projection matrix stack.

GL_TEXTURE

Applies subsequent matrix operations to the texture matrix stack.

GL_COLOR

Applies subsequent matrix operations to the color matrix stack.

27)gluPerspective — set up a perspective projection matrix

C Specification

void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar); Parameters for y Specifies the field of view angle, in degrees, in the y direction .aspect Specifies the aspect ratio that determines the field of view in the x direction.

The aspect ratio is the ratio of x (width) to y (height).

zNear

Specifies the distance from the viewer to the near clipping plane

(always positive).

zFar

Specifies the distance from the viewer to the far clipping plane (always positive).

Description

gluPerspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in gluPerspective should match the aspect ratio of the associated viewport. For example,aspect= 2.0 means the viewer's angle of view is twice as wide in x as it is in y. If the viewport is twice as wide as it is tall, it displays the image without distortion.

28)glLoadIdentity — replace the current matrix with the identity matrix

C Specification

void glLoadIdentity(void);

Description

glLoadIdentity replaces the current matrix with the identity matrix.It is semantically equivalent to calling glLoadMatrix with the identity matrix

29)glTranslate — multiply the current matrix by a translation matrix

C Specification

void glTranslated(GLdouble x, GLdouble y, GLdouble z);

void glTranslatef(GLfloat x, GLfloat y, GLfloat z);

Parameters x, y, z

Specify the x, y, and z coordinates of a translation vector.

Description

glTranslate produces a translation by (x,y, z)

The current matrix (see glMatrixMode) is multiplied by this translation matrix, with the product replacing the current matrix, as if glMultMatrix were called with the following matrix for its argument:

30)glutSolidSphere, glutWireSphere

glutSolidSphere and glutWireSphere render a solid or wireframe sphere respectively.

Usage

void glutSolidSphere(GLdouble radius,GLint slices, GLint stacks);

void glutWireSphere(GLdouble radius,GLint slices, GLint stacks);

radius - The radius of the sphere.

Slices - The number of subdivisions around the Z axis (similar to lines of longitude).

Stacks - The number of subdivisions along the Z axis (similar to lines of latitude).

Description

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

31)glutSwapBuffers

glutSwapBuffers swaps the buffers of the current window if double buffered.

Usage

void glutSwapBuffers(void);

Description

Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

Steps to install OpenGL-

1) Download Dev C++ and perform installation steps.
2) Type freeglut in search engine ---- > click on download ----- > go to MSVC & MinGW folder ----- > download **freeglut 3.0.0 MinGW package** ----- > extract the folder
3) Go to folder **C:\Program files(x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32\include\GL**
4) In another window open **freeglut\include\GL** then copy all files and paste in GL folder of Dev-Cpp
5) Go to folder **freeglut\lib\x64** then copy all files and paste in Dev-Cpp lib folder
6) Go to folder **freeglut\bin\x64** then copy all files and paste in **C:\\Windows\System32**
7) Open Dev-Cpp application then go to file --- > new project --- > console application ---- > give project name --- >click on OK
8) Right click on project name ---- > project option --- > parameter ---- > type in linker
   -lopengl32
   -lfreeglut
   -lglu32

9) Then click on OK.

Conclusion:

Successfully install the OpenGL .

Oral Questions:

1. What is OpenGL.?
2. What are the different functions and features of OpenGL?
3. What is the use of GL_TRIANGLES?

4. What are the header files include in OpenGL program?
5. Which software is use for OpenGL programming?

Assignment No. 2

Problem Statement:

Implement DDA and Bresenham line drawing algorithm to draw: i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line ; using mouse interface Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes positive as well as negative.

Aim:-

To learn Digital Differential Analyser (DDA) Line Drawing Algorithm.

Prerequisite:

Basic primitives of computer graphics and concepts of OOP.

Theory:

Line:

It is the path between two end points. Any point (x, y) on the line must follow the line equation: Point is the fundamental element of the picture representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending whether the data are two or three dimensional. Thus (x1, y1) or (x1, y1, z1) would represent a point two or three dimensional space.

Two points used to specify line by below equation y = m * x + b, where – m is the slope of the line. – b is a constant that represent the intercept on y-axis.

If we have two end points, (x0, y0) and (x1, y1), then the slope of the line (m) between those points can be calculated as: m = Δy / Δx = (y1 – y0) / (x1 – x0) Draw a line with the help of line equation is very time consuming because it’s required lots of calculation. A cathode ray tube (CRT) raster display is considered a matrix of discrete finite area cells (pixels), it is not possible to draw a straight line directly from one point to another. The process of determining which pixels provide the best approximation to the desired line is called rasterization.

To draw a line, we can use two algorithms:

- ❖ **DDA (Digital Differential Analyzer)/Vector Generation Algorithm.**
- ❖ **Bresenham’s Line Algorithm.**

1) DDA Line Drawing Algorithm:

- DDA is an incremental scan conversion method to determine points on screen to draw a line where the Start and End coordinates of the Line segment are provided.
- DDA calculates the length of the line segment with respect to the difference between either X coordinates or Y coordinates, whichever is greater.
- In DDA, we either step across X-Direction and solve for Y (In case of gentle slope lines) or we step Y-Direction and solve for X (incase of sharp slope lines) with the help of increment in either X and/or Y directions.

- As the increments are calculated with respect to X or Y direction ,so one of the increment will be either (1,/0/-1) and the other increment may be in float.
- Floating point arithmetic in DDA algorithm is time-consuming which results in poor end point accuracy.
- It is the simplest algorithm and does not require special skills for implementation.

DDA Algorithm:

```
1)  Read the line endpoints (X1, Y1) and (X2, Y2) such that they are not equal.

2)  Calculate
dx  = X2 – X1
dy  =  Y2 –Y1

3)  if (dx >dy) then
step = dx
else
step = dy
end if

4)  Xinc =  dx / step
    Yinc = dx / step

5)  putpixel (X1, Y1,1)
    x = X1
    y = Y1

6)  i = 1

    while (i <= step)
     {
    putpixel( x, ,y 1)
        x = x + Xinc
        y = y + Yinc
        i =  i + 1
    }

7)  Stop.
```

Advantages of DDA

- Simplest algorithms to implement.
- Faster method for calulating pixel positions than direct use of eaquation y=mx+b.it eleminates the multiplication by making use of raster characteristics

Disadvantages of DDA

- Floating point arithmetic in DDA is time consuming
- The algorithm is orientation dependent.hence end point accuracy is poor.

2) Bresenham's line drawing algorithm:

The Bresenham's algorithm is another incremental scan conversion algorithm.An accurate and efficient raster line-generating algorithm, developed by Bresenham, scans converts lines using only incremental integer calculations that can be adapted to display circles and other curves .The big advantage of this algorithm is that it uses only integer calculationsMove across the *x* axis in unit intervals and at each step choose between two different *y* coordinates.

Principle:

Either x or y is incremented by one unit depending on slope of line and increment in other variable is determined by checking the distance between true line segment and nearest pixel.
This distance is called as **decision variable.**

Initially this decision variable s set as
e = 2 * dy - dx.

Algorithm:

1) Read the line endpoints (X1, Y1) and (X2, Y2) such that they are not equal.

2) Calculate
   dx = X2 – X1

```
      dy = Y2 – Y1

   3)  Initialize
              x = X1
             y = Y1

   4)  Calculate  decision variable
             e = 2 * dy – dx

   5)  i = 1

      putpixel ( x, y,1)

   6)   while ( e >= 0)
          {
              y = y + 1
              e = e – 2 * dx
           }

              x = x + 1
              e = e + 2 * dy

   7)  i = 1

   8)  if ( i <= dx) then  go to step
9)Stop.
```

Conclusion:-

In This way we have studied that how to draw a line using DDA and Bresenham’s Algorithms.

Oral Questions:-

Q 1: Which algorithm is faster in DDA and Bresenham’s?

Q 2: State advantages of Bresenham’s Line drawing algorithm?

Q 3: Difference between DDA Line and Bresenham’s line algorithm?

Q 4: Which Algorithm is efficient for line drawing?

Q 5: What is long form of DDA?

Assignment No. 3

Problem Statement:

Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius

Aim:-

To draw Circle using Bresenham’s algorithm

Prerequisite:

Basic of Circle Generation algorithms.

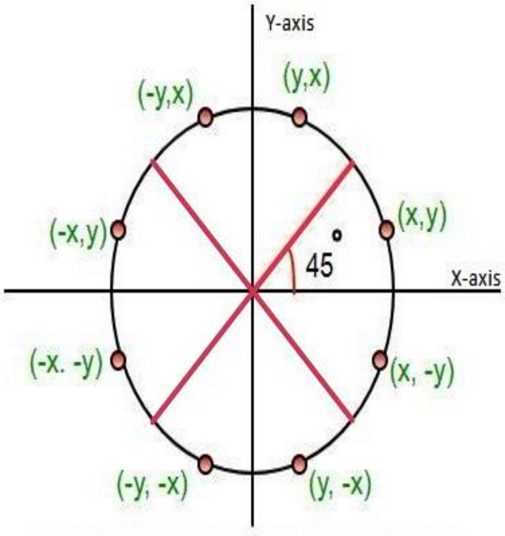
Theory:

Bresenham’s Circle Drawing Algorithm is a circle drawing algorithm that selects the nearest pixel position to complete the arc. The unique part of this algorithm is that is uses only integer arithmetic which makes it, significantly, faster than other algorithms using floating point arithmetic in classical processors.

Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.

- We know that there are 360 degrees in a circle. First we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated.
- Next we see that it's also symmetrical about the y axis, so now we only need to calculate the first 90 degrees.
- Finally we see that the circle is also symmetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees.
- We only need to calculate the values on the border of the circle in the first octant. The other values may be determined by symmetry.

Bresenham's circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x, y) it calculates, we draw a pixel in each of the eight octants of the circle. This is done till when the value of the y coordinate equals the x coordinate. The pixel positions for determining symmetry are given in the below algorithm.



For each pixel (x,y) all possible pixels in 8 octants

Here the equation for di at starting point, i.e. at x=0 and y=r can be simplified as,

di=3-2r (where r is radius)

For di <0, xi+1 = xi + 1 and

For di >=0, xi+1 = xi +1 and yi+1 = yi –1

Similarly, the equations for di+1 for both the cases are given as

For di <0, di +1 = di + 4xi+6 and

For di >=0, di +1 = di + 4(xi-yi) +10.

Reflecting about x & y-axis can draw the remaining part of the circle.

Algorithm:

```
1. Read the radius(r) of the circle.
2. d=3-2r (Initialize the decision variable)
3. x=0, y=r    ( Initialize starting point)
4. do
 {
        plot(x,y)
        plot(y,x)
        plot(y,-x)
        plot(x,-y)
        plot(-x,-y)
        plot(-y,-x)
        plot(-y,x)
        plot(-x,y)
   if(d<0) then
 {
   d=d+4x+6
 }
  else
 {
 d=d+4(x-y)+10
y=y-1
}
x=x+1
}while(x<y)
```

5. Stop.

Advantages of Bresenham Circle Drawing Algorithm-

- The entire algorithm is based on the simple equation of circle $X^2 + Y^2 = R^2$.
- It is easy to implement.

Disadvantages of Bresenham Circle Drawing Algorithm-

- accuracy of the generating points is an issue in this algorithm.
- This algorithm suffers when used to generate complex and high graphical images.
- There is no significant enhancement with respect to performance.

Conclusion:-

In This way we have studied that how to draw a circle on the screen

Oral Questions:-

Q 1: Explain the Bresenhams circle drawing algorithm?

Q 2: Explain the circle equation.

Q.3: What is the value of decision parameter in Bresenham circle drawing algorithm?

Assignment No. 4

Problem Statement:

Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming

Aim:-

To fill polygon using different polygon filling methods.

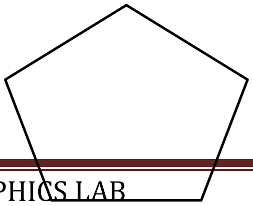
Prerequisite:

Basic of polygon types and polygon filling algorithms.

Theory:

Polygon

When starting and ending point of any polyline is same i.e. when polyline is closed then it is called polygon

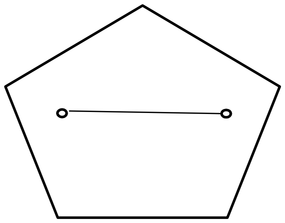


There are two types of polygon

The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie.

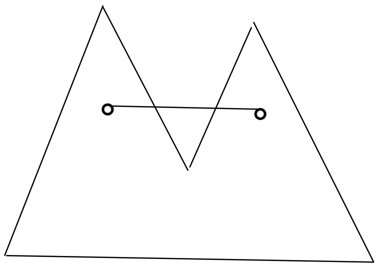
1) Convex polygon

It is a polygon in which any two points taken are surely inside the polygon lies complete inside the polygon, then that polygon is called as convex polygon.



2)Concave polygon

It is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.



3)Convex polygon

A convex polygon is defined as a polygon with all its interior angles less than 180°. This means that all the vertices of the polygon will point outwards, away from the interior of the shape. Think of it as a 'bulging' polygon. Note that a triangle (3-gon) is always convex.

Polygon filling

It is the process of coloring in a fixed area or region. It also means high lighting all the pixels which lie inside the polygon with any color then other background color. There are two basic approaches used to fill the polygon. One way to fill polygon is to start from a given “seed “point known to be inside the polygon & highlight outward from the point in neighboring pixels. Until we encounter the boundary pixel. The approach is called “**seed fill.”** Another approach to fill polygon is “**scan line”** in which it is checked whether pixel is inside the polygon or outside the polygon.

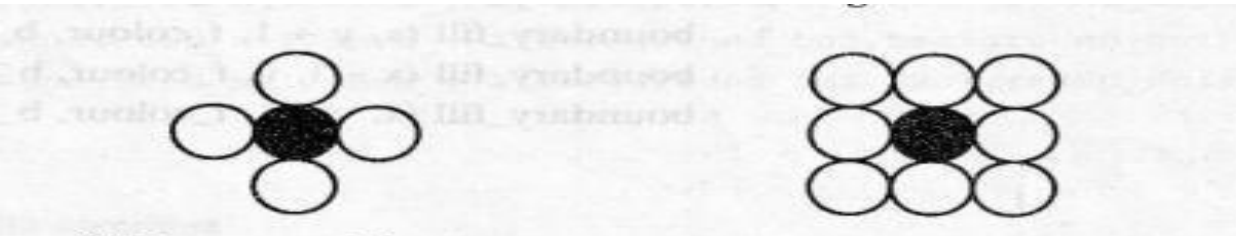
Seed Fill

The seed fill algorithm is further classified as flood algorithm and boundary fill algorithm. Algorithms that fill interior defined regions are called flood-fill algorithms; those that fill boundary defined regions are called boundary fill algorithms or edge fill algorithms.

Boundary fill algorithm

In this algorithm, one point which is surely inside the polygon is taken. This point is called seed point which is nothing but a point from which filling process starts. Boundary defines region may be either 4-connected or 8-connected.

Check the color of pixel If boundary color i.e pixel are not reached, pixels are highlighted (by fill colored) & the process is continued until boundary pixels are reached.



(a) Four connected region (b) Eight connected region

Algorithm

```
Boundary_fill(x,y,f_color,b_color)
{
if(getpixel(x,y)!=b_color &&getpixel(x,y)!=f_color)
{
putpixel(x,y,f_color)
Boundary_fill(x+1,y,f_color,
b_color)
Boundary_fill(x,y,+1f_color,
b_color) Boundary_fill(x-
1,y,f_color,b_color)
Boundary_fill(x,y-
1,f_color,b_color)
}
}
```

Flood Fill Algorithm

In this algorithm, areas are filled by replacing a specified interior color instead of searching a specified interior boundary color. Here, instead of checking boundary color, whether pixels are having polygons original color is checked. Using either 4-connected or 8-connected approach step through pixel position until all interior point has been filled.

Algorithm

```
Flood_fill (x,y, old_color, new_color)
{
If (getpixel(x,y)=old_color)
{
Putpixel (x,y,new color)
Flood_fill (x+1,y, old_color, new_color)
Flood_fill (x-1,y, old_color, new_color)
Flood_fill (x,y,+1 old_color, new_color)
Flood_fill (x,y-1, old_color, new_color)
Flood_fill (x+1,y+1, old_color, new_color)
Flood_fill (x-1,y-1, old_color, new_color)
Flood_fill (x+1,y-1, old_color, new_color)
Flood_fill (x-1,y+1, old_color, new_color)
}
}
```

Hence, getpixel function gives color of specified pixel & putpixel function draws pixel specified color

Conclusion:-

In This way we have studied that how to draw a convex polygon and how to fill a polygon with seed fill algorithm.

Oral Questions:-

Q.1: Define polygon?

Q.2: What are the types of polygon?

Q.3: Explain convex polygon?

Q.4: Define Concave polygon.

Q.5: How many ways to represent polygon?

Q.6: How many methods of polygon filing?

Assignment No. 5

Problem Statement:

Implement Cohen Sutherland polygon clipping method to clip the polygon with respect the viewport and window. Use mouse click, keyboard interface.

Aim:-

To implement Cohen Sutherland polygon clipping method.

Prerequisite:

Basic of viewport,window and polygon clipping algorithms.

Theory:

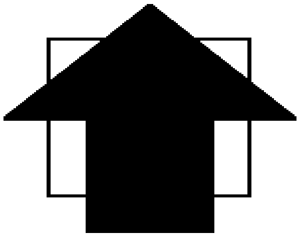
Polygon Clipping:-

Polygon clipping is a process in which we only consider the part which is inside the view pane or window. We will remove or clip the part that is outside the window. We will use the following algorithms for polygon clipping-

1. **Sutherland-Hodgeman polygon clipping algorithm**
2. **Weiler-Atherton polygon clipping algorithm**

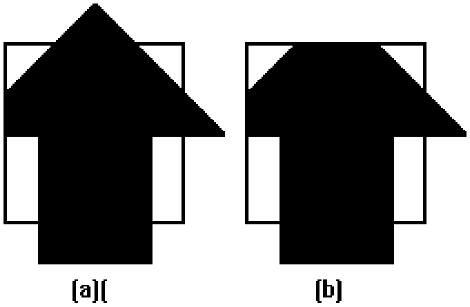
Sutherland-Hodgeman polygon clipping algorithm

The Sutherland - Hodgman algorithm performs a clipping of a polygon against each window edge in turn. It accepts an ordered sequence of verices v1, v2, v3, ..., vn and puts out a set of vertices defining the clipped polygon.



Before clipping

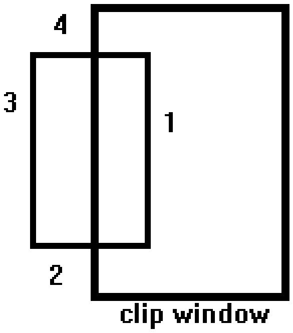
This figure represents a polygon (the large, solid, upward pointing arrow) before clipping has occurred.



The following figures show how this algorithm works at each edge, clipping the polygon.

a) Clipping against the left side of the clip window.
b) Clipping against the top side of the clip window.
c) Clipping against the right side of the clip window.
d) Clipping against the bottom side of the clip window.

As the algorithm goes around the edges of the window, clipping the polygon, it encounters four types of edges. All four edge types are illustrated by the polygon in the following figure. For each edge type, zero, one, or two vertices are added to the output list of vertices that define the clipped polygon.



The four types of edges are:

1. Edges that are totally inside the clip window. - add the second inside vertex point
2. Edges that are leaving the clip window. - add the intersection point as a vertex
3. Edges that are entirely outside the clip window. - add nothing to the vertex output list

Edges that are entering the clip window. - save the intersection and inside points as vertices

Sutherland-Hodgeman Polygon Clipping Algorithm:-

1. Read coordinates of all vertices of the Polygon.
2. Read coordinates of the dipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane.
5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary.
6. Repeat the steps 4 and 5 for remaining edges or the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

The Advantages of Sutherland-Hodgman algorithm are:

- It is very useful for clipping polygons. ...
- It is easy to implement.
- It works by extending each line of the convex clip polygon. ...
- It selects only those vertices which are on the visible side of the subject polygon.

The Disadvantages of Sutherland-Hodgman algorithm are:

- Convex polygons are correctly clipped by the Sutherland-Hodgeman Algorithm. But, **concave polygons cannot be clipped correctly**.
- It may be displayed with extraneous lines.
- It clips to each window boundary one at a time.

- **Conclusion:-**

we have studied that how to clip polygon using cohen Sutherland algorithm.The Sutherland - Hodgman algorithm performs a clipping of a polygon against each window edge in turn. It accepts an ordered sequence of verices v1, v2, v3, ..., vn and puts out a set of vertices defining the clipped polygon.

Oral Questions:-

Q.1: Explain the algorithm for Line clipping?

Q.2: What is the difference between Line clipping and Polygon clipping?

Q.3: . How to find intersection point of any edge of window and line segment?

Q.4: What do you mean by homogeneous coordinates?

Q.5: Where we can use homogeneous coordinates?

Q.6: . Explain Sutherland-hodgeman polygon clipping algorithm?

Assignment No. 6

Problem Statement:

Implement following 2D transformations on the object with respect to axis :

i) Scaling ii) Rotation about arbitrary point iii) Reflection

Aim:-

To draw 2-D object and perform basic transformations

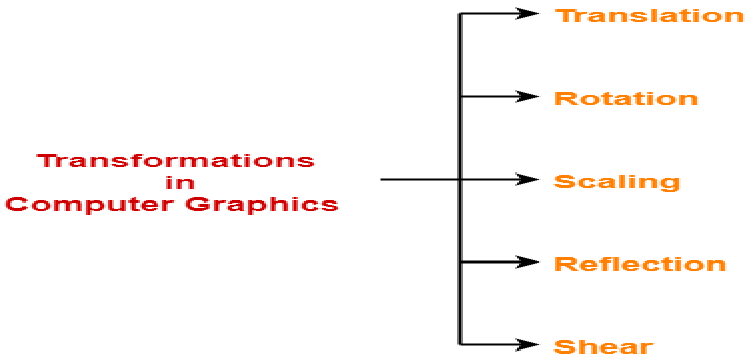
Prerequisite:

Basic of 2D transformations.

Theory:

2D Transformations in Computer Graphics-

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.



1) Scaling

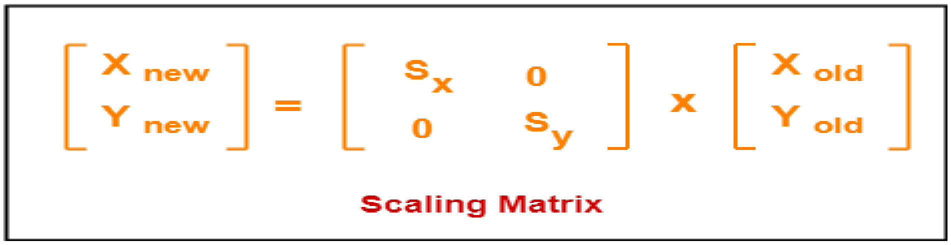
To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (X, Y), the scaling factors are (SX, SY), and the produced coordinates are (X', Y'). This can be mathematically represented as shown below −

X' = X . SX

Y' = Y . SY

The scaling factor SX, SY scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below –



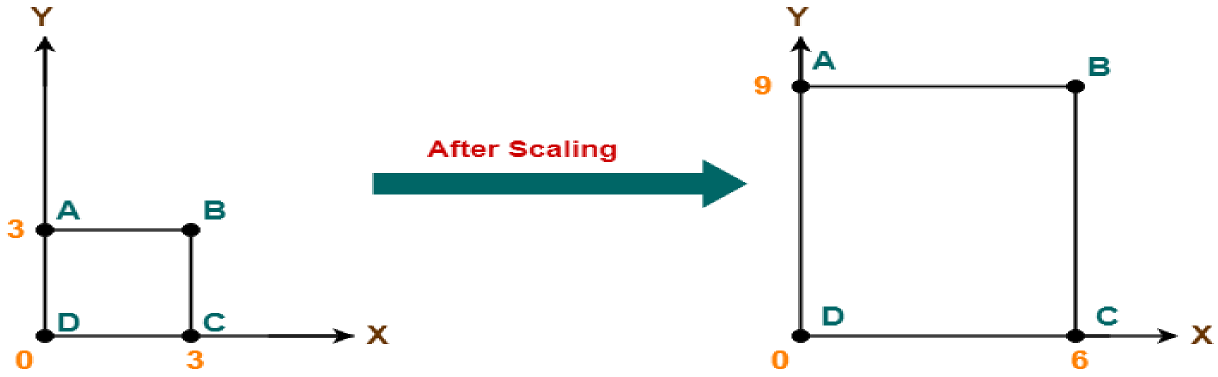
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Scaling Matrix

OR

P' = P . S

Where S is the scaling matrix. The scaling process is shown in the following figure



If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

2)Rotation about an arbitrary point

If we want to rotate an object or point about an arbitrary point, first of all, we translate the point about which we want to rotate to the origin. Then rotate point or object about the origin, and at the end, we again translate it to the original place. We get rotation about an arbitrary point.

Suppose the reference point of rotation is other than origin, then in that case we have to follow series of transformation.

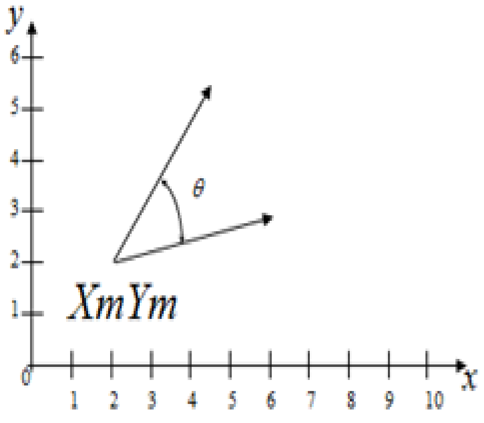


Figure 17

Consider figure 17, assume that we have to rotate a point P1 with respect to (Xm, Ym) then we have to perform three steps.

I) **Translation:**

First we have to translate the (Xm, Ym) to origin as shown in figure 18. Translation matrix (T1) will become

$$T1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xm & -Ym & 1 \end{bmatrix} \Rightarrow$$

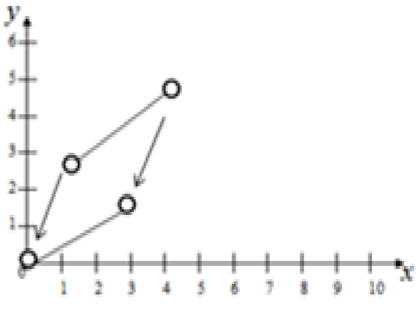


Figure 18

Here in figure 18, Tx = -Xm and Ty = -Ym.

II) **Rotation:** Rotate it in clockwise or anticlockwise direction. See figure 19. Let assume as anticlockwise rotation by angle θ. So our rotation matrix will be

$$R = \begin{bmatrix} Cos\theta & Sin\theta & 0 \\ -Sin\theta & Cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow$$

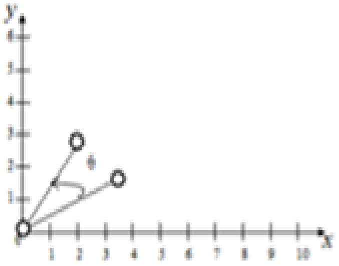


Figure 19

III) **Translation:** Translate back to original position. So the translation matrix T2 will become

$$T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xm & Ym & 1 \end{bmatrix} \Rightarrow$$

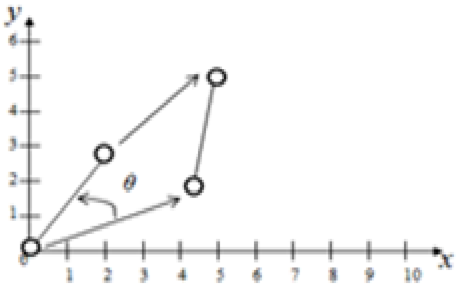


Figure 20

Now let us form a combined matrix.

M = Translation * Rotation * Translation

= T1 * R * T2

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xm & -Ym & 1 \end{bmatrix} * \begin{bmatrix} Cos\theta & Sin\theta & 0 \\ -Sin\theta & Cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xm & Ym & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xm & -Ym & 1 \end{bmatrix} * \begin{bmatrix} Cos\theta & Sin\theta & 0 \\ -Sin\theta & Cos\theta & 0 \\ Xm & Ym & 1 \end{bmatrix}$$

$$= \begin{bmatrix} Cos\theta & Sin\theta & 0 \\ -Sin\theta & Cos\theta & 0 \\ -Xm * Cos\theta + Ym * Sin\theta + Xm & -Xm * Sin\theta - Ym * Cos\theta + Ym & 1 \end{bmatrix}$$

This transformation matrix is the overall transformation matrix for rotation about arbitrary point (Xm, Ym) by an angle θ in anticlockwise direction.

3)Reflection

- Reflection is a kind of rotation where the angle of rotation is 180 degree.
- The reflected object is always formed on the other side of mirror.
- The size of reflected object is same as the size of original object.

Consider a point object O has to be reflected in a 2D plane.

Let-

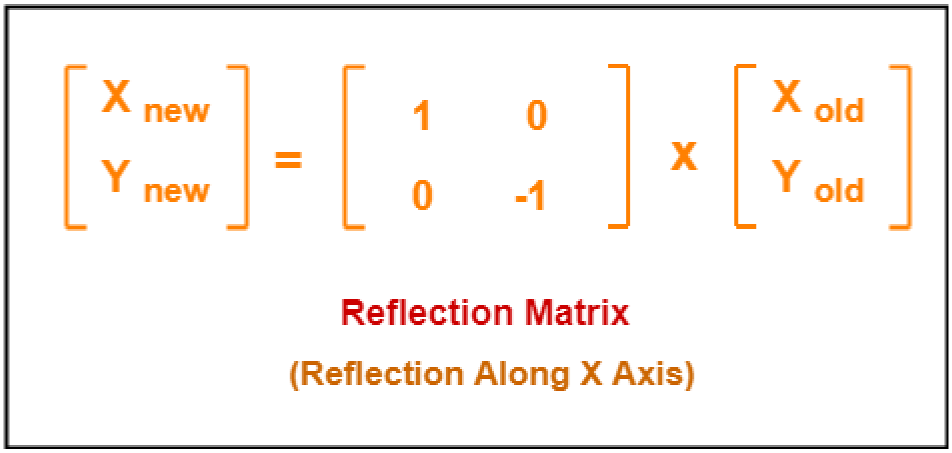
- Initial coordinates of the object O = (X_{old}, Y_{old})
- New coordinates of the reflected object O after reflection = (X_{new}, Y_{new})

Reflection On X-Axis:

This reflection is achieved by using the following reflection equations-

- $X_{new} = X_{old}$
- $Y_{new} = -Y_{old}$

In Matrix form, the above reflection equations may be represented as-



$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Reflection Matrix

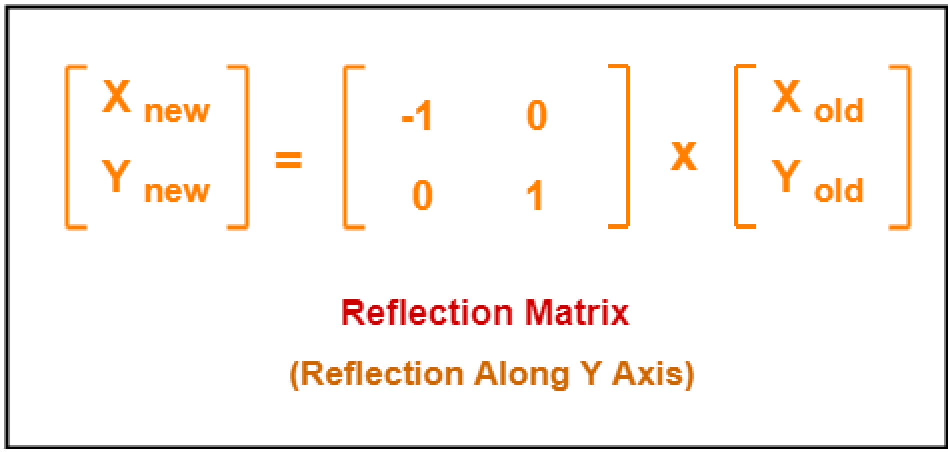
(Reflection Along X Axis)

Reflection On Y-Axis:

This reflection is achieved by using the following reflection equations-

- $X_{new} = -X_{old}$
- $Y_{new} = Y_{old}$

In Matrix form, the above reflection equations may be represented as-



$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Reflection Matrix

(Reflection Along Y Axis)

Conclusion:-

In This way we have studied that how to transform a given polygon in 2D.

Oral Questions:-

Q.1: State and explain different 2D transformation?

Q.2: Explain the concept of Homogeneous coordinates?

Q.3: Explain 2D Translation & Scaling?

Q.4: Explain the clockwise and anticlockwise 2D rotation?

Assignment No. 7

Problem Statement:

Generate fractal patterns using i) Bezier ii) Koch Curve .

Aim:-

To Generate fractal patterns using Bezier and Koch Curve .

Prerequisite:

Basic of fractal and its types.

Theory:

Bezier Curve-

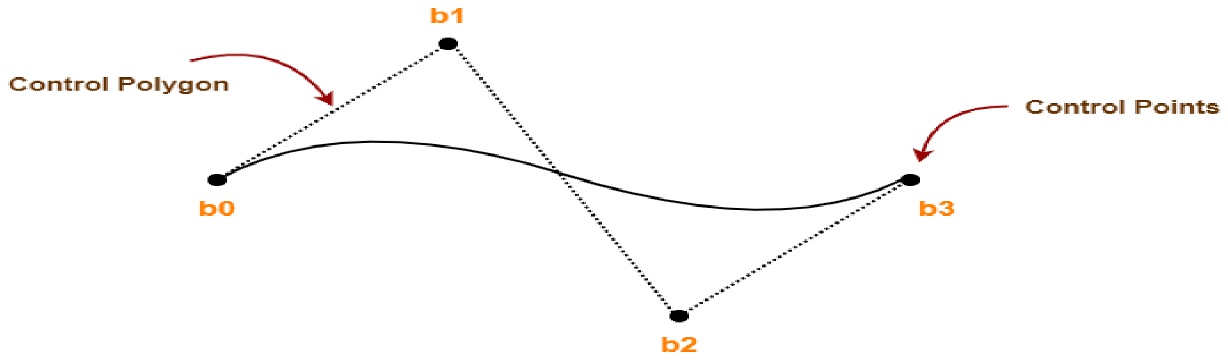
Bezier Curve may be defined as-

- Bezier Curve is parametric curve defined by a set of control points.
- Two points are ends of the curve.
- Other points determine the shape of the curve.

The concept of bezier curves was given by Pierre Bezier.

Bezier Curve Example-

The following curve is an example of a bezier curve-



Bezier Curve Example

Here,

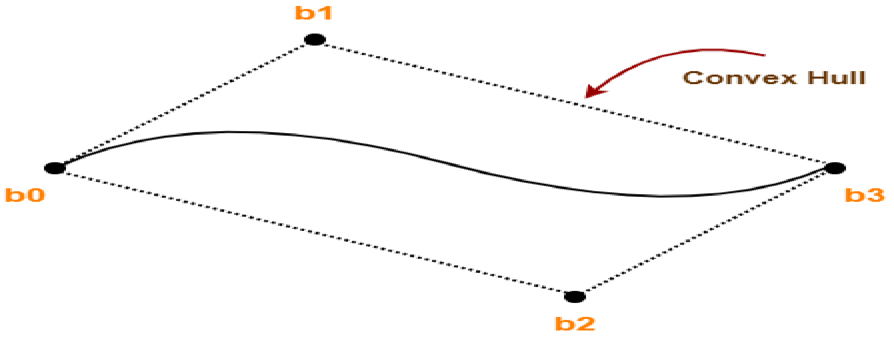
- This bezier curve is defined by a set of control points b_0, b_1, b_2 and b_3.
- Points b_0 and b_3 are ends of the curve.
- Points b_1 and b_2 determine the shape of the curve.

Bezier Curve Properties-

Few important properties of a bezier curve are-

Property-01:

Bezier curve is always contained within a polygon called as convex hull of its control points.



Bezier Curve With Convex Hull

Property-02:

- Bezier curve generally follows the shape of its defining polygon.

- The first and last points of the curve are coincident with the first and last points of the defining polygon.

Property-03:

The degree of the polynomial defining the curve segment is one less than the total number of control points

Degree = Number of Control Points – 1

Property-04:

The order of the polynomial defining the curve segment is equal to the total number of control points.

Property-05:

- Bezier curve exhibits the variation diminishing property.
- It means the curve do not oscillate about any straight line more often than the defining polygon

Bezier Curve Equation-

A bezier curve is parametrically represented by-

$$\boxed{P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t)}$$

Bezier Curve Equation

Here,

- t is any parameter where 0 <= t <= 1
- P(t) = Any point lying on the bezier curve
- B_i = ith control point of the bezier curve
- n = degree of the curve

- $J_{n,i}(t)$ = Blending function = $C(n,i)t^{i}(1-t)^{n-i}$ where $C(n,i) = n! / i!(n-i)!$

Applications of Bezier Curves-

Bezier curves have their applications in the following fields-

1. Computer Graphics-

- Bezier curves are widely used in computer graphics to model smooth curves.
- The curve is completely contained in the convex hull of its control points.
- So, the points can be graphically displayed & used to manipulate the curve intuitively.

2. Animation-

- Bezier curves are used to outline movement in animation applications such as Adobe Flash and synfig.
- Users outline the wanted path in bezier curves.
- The application creates the needed frames for the object to move along the path.
- For 3D animation, bezier curves are often used to define 3D paths as well as 2D curves.

3. Fonts-

- True type fonts use composite bezier curves composed of quadratic bezier curves.
- Modern imaging systems like postscript, asymptote etc use composite bezier curves composed of cubic bezier curves for drawing curved shapes.

Koch Curve:-

- The Koch curve can be drawn by dividing line into 4 equal segments with scaling factor 1/3 and middle two segments are so adjusted that they form adjacent sides of an equilateral triangle as shown in the Fig. 16(a) .This is the first approximation to the koch curve.
- To apply the second approximation to the Koch curve we have to repeat the above process for each of the four segments. The resultant curve is shown in Fig. 16(b).

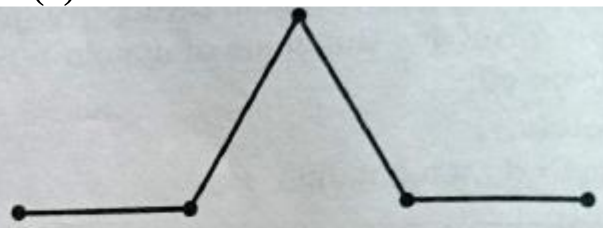


Fig. 16(a) First approximation of the koch curve

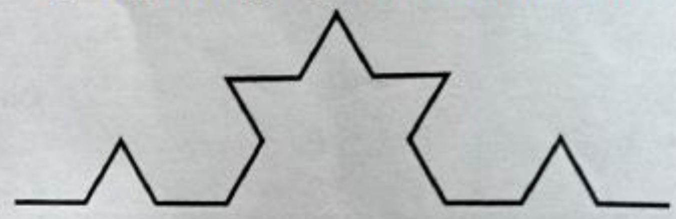


Fig. 16(b) The second approximation to koch curve

The resultant curve has more wiggles and its length is 16/9 times the original length.

From the above figures we can easily note following points about the koch curve :

- Each repetition increases the length of the curve by factor 4/3.
- Length of curve is infinite.
- Unlike Hibert's curve, it doesn’t fill an area.
- It doesn’t deviate much from its original shape.
- If we reduce the scale of the curve by 3 we find the curve that looks just like the original one; but we must assemble 4 such curves to make the originals, so we have

$$4 = 3^{D}$$

Solving for D we get

$$D = \log_3 4 = \log 4 / \log 3 = 1.2618$$

Therefore for koch curve topological dimension is 1 but fractal dimension is 1.2618.

From the above discussion we can say that point sets, curves and surfaces which give a fractal dimension greater than the topological dimension are called fractals The Hilbert's curve and koch curves are fractals, because their fractal dimensions (respectively, 2 and 1.2618) are greater than their topological dimension which is 1.

Successive Refinement or Koch curve:-

The Koch Curve Fractal:-

A complex curve can be constructed by repeatedly refining a simple curve. successive generations of the Koch curve are denoted K0, KI, K2….. . The zeroth generation shape K0 is a horizontal line of length unity.

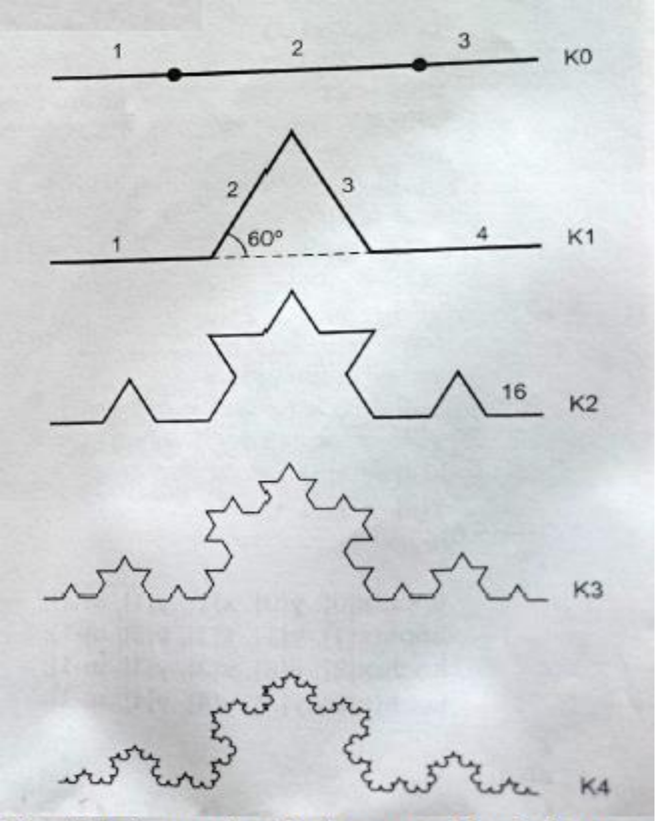


Fig. (17) Successive Refinement of koch Curve

The Koch curve is a fractal that starts with a simple pattern made of a line that is divided into 3 equal parts.

Erase the middle segment and replace it with an upsidedown “V” shape, and now the whole pattern is made up of four line segments. The total length of the line is evidently 4/3.

Next, we do the same thing again. Each of those four lines is divided in thirds, and the middle segment is replaced with a “V”. There are now 4 x 4 or 16 line segments.

Fractals are never-ending patterns made by repeating the same idea over again. So next, we'll replace each of the 16 line segments with the same pattern again.

Each time we do this. the curve gets more jagged and complicated, and its length - or perimeter - gets bigger. The K1 curve has a total length of (4/3)i.

Eventually, the pattern starts to look like a fractal in nature, such as a coastline, or part of a snowflake.

Conclusion:-

In This way we have studied that how to Generate fractal patterns using Bezier and Koch Curve .

Oral Questions:-

Q.1: Define fractal.

Q.2: Explain application of Bezier curve.

Q.3: Explain Koch curve .

Assignment No. 8

Problem Statement:

Implement animation principles for any object.

Aim:-

To implement animation principles for any object .

Prerequisite:

Basic of animation and its principles.

Theory:

In todays world which is filled with full of imagination and visualizations, there are some areas which is covered with the word animation. When this word will come in anyone’s mind they always create a picture of cartoons and some Disney world shows. As we already know that among kids, animation movies are very popular like Disney world, Doraemon, etc. All the cartoons and animation pictures are type of animation which made from a thousands of single pictures add together and play according to steps.

Animation is defined as a series of images rapidly changing to create an illusion of movement. We replace the previous image with a new image which is a little bit shifted. Animation Industry is having a huge market nowadays. To make an efficacious animation there are some principles to be followed.

Types of animation:

- Traditional Animation
- 2D Animation

- 3D Animation
- Motion Graphics
- Stop Motion

1. Traditional Animation

Traditional animation can also be referred to as cell animation. This type of animation requires the animator to draw every single frame by hand to create an animated scene. This is usually done on a light table that allows the artists to see the previous drawing through the top layer of paper. Well-known companies like Disney are known for using this type of animation. Traditional animation is still done today on computers with special tablets.

2. 2D Animation

2D animation refers to vector-based animations similar to the ones used in Flash. This style of animation has been growing in popularity because the technology is so accessible. Although artists have the option of editing frame by frame, vector-based animation gives the artist the option to create rigs for the characters and move single body parts at a time rather than constantly redrawing the characters. It gives more flexibility to beginners in animation because they don’t have to rely so heavily on drawing skills.

3. 3D Animation

3D animation is also known as computer animation and it is currently the most commonly used form of animation. The process of 3D animation is very different from the traditional style but they both require the artist to share the same principles of movement and composition in animation. 3D animation has less to do with drawing and more to do with moving a character in a program. The National Science Foundation emphasizes how heavily 3D animators must rely on physics to create realistic animations. The animator creates keyframes or specific movements and lets the computer fill in the rest.

4. Motion Graphics

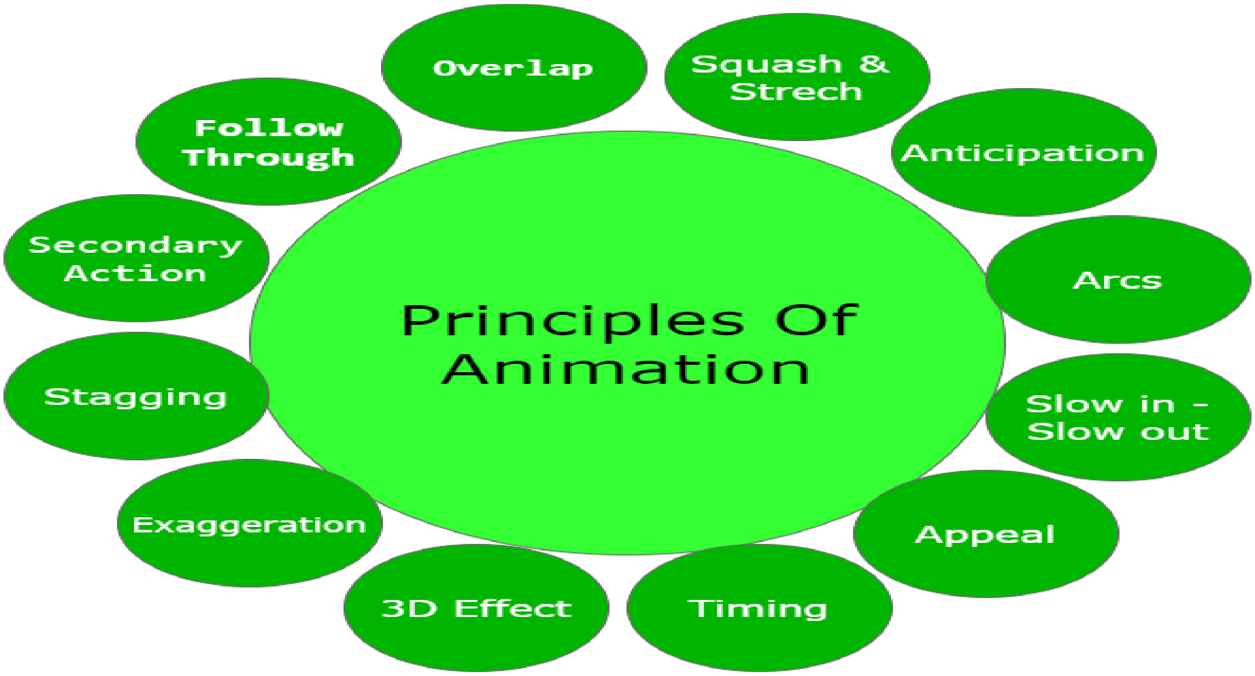
Unlike the previously mentioned types of animation, motion graphics are not driven by characters or storylines. This art form focuses on the ability to move graphic elements, shapes, and text. This process is commonly used for things like television promotions, explainer videos, and animated logos. The skillset necessary for the other types of animation doesn't apply to motion graphics because there's no need to mimic body movement or facial expressions. Advertisements rely heavily on motion graphics and present plenty of career opportunities.

5. Stop Motion

Stop motion animation is very similar to traditional animation because it combines a series of still images that are slightly different to show movement. The largest difference is that stop motion uses photography and captures real objects. With stop motion, the artists take a photo of an object or scene and slightly moves the objects before taking another photo. The artist repeats this process until the scene is completed and uses each photo as a frame in the animation. It's similar to a flipbook with photos.

Principle of Animation:

There are 12 major principles for an effective and easy to communicate animation.



1. **Squash and Stretch –**
   This is the most important principle of animation, it gives the sense of weight and volume to draw an object.

2. **Anticipation –**
   In this principle animator will create a starting scene like that it shows that something will happen, almost nothing happens suddenly.

3. **Staging –**
   Animator creates such type of scene which attract audience so that audience's attention is directed toward that scene.

4. **Straight Ahead –**
   In this principle, all frames are drawn from beginning to the end and then fill all the interval or scene.

5. **Flow through and overlapping action –**
   Two object's action have different speed in any scene can easily describe this principle.

6. **Slow in and Slow out –**
   When an abject have maximum acceleration in between and resist on the beginning and end will show this principle's working.

7. **Arc –**
   Arcs are present in almost all animation as no object will follow straight line and follows some arc in its action.

8. **Secondary action –**
   As with one character's action second character move shows the multiple dimension of an animation.

9. **Timing –**
   For playing a given action a perfect timing is very important.

10. **Exaggeration –**
This principle creates extra reality in the scene by developing a proper animation style.

11. **Solid drawing –**
In this principle, any object will created into 3D form to get realistic visualization of scene.

12. **Appeal –**
Any character need not be as same as any real character but it somewhat seems to be like that which create a proper thinking in the audience's mind.

Conclusion:-

In This way we have studied principles of animation.

Oral Questions:-

Q.1: What is animation?

Q.2: Explain different types of animation.

Q.3: What are the principles of animation?